**Android播放器sdk**

**概述**

AlivcMediaPlayer是一款基于Android平台的多媒体视频播放SDK。它为IOS的开发者提供了简单易用的接口，帮助开发者方便快捷、低门槛的实现多媒体播放功能的开发。它支持HLS、RTMP、HTTP FLV、MP4等多种流媒体播放格式，视频支持h264格式、音频支持AAC格式。另外，针对直播用户的需求，还增加了首帧秒开的功能；同时为了减少直播的延迟，增加了弱网条件下播放的跳帧功能。

**版本和新增功能**

| **功能** | **版本** |
| --- | --- |
| 支持HLS、RTMP、HTTP FLV、mp4等流格式 | v1.0 |
| 支持h264+aac | v1.0 |
| 支持armv7、arm64 | v2.0 |
| 支持直播首帧秒开 | v2.1 |
| 支持弱网条件下的丢帧策略 | v2.1 |
| 支持多实例，支持https | v2.2 |
| 支持带切边的视频渲染模式 | v2.2 |

**阅读对象**

本文档面向所有使用该SDK的开发人员、测试人员以及对此感兴趣的用户，要求开发者对播放器的基本功能有一定的了解。

**开发准备**

**设备和系统版本**

android4.0及以上

手机芯片要求armv7或armv8架构

**安装包下载及说明**

安装包的下载地址为：点击下载

播放器SDK的完整下载包中包含demo、doc、lib等:

1. demo：主要存放了调用SDK的示例工程，可以帮助用户了解如何使用该SDK。
2. lib：播放器SDK开发包,包括jar文件和so文件。
3. doc：存放SDK相关接口文档。

**快速开发**

**开发环境配置**

1. 需要配置好maven的Android开发环境。
2. 在阿里云官网上注册云帐号，并开通视频点播或视频直播服务。方法如下：

[视频点播服务开通](https://help.aliyun.com/document_detail/29938.html)

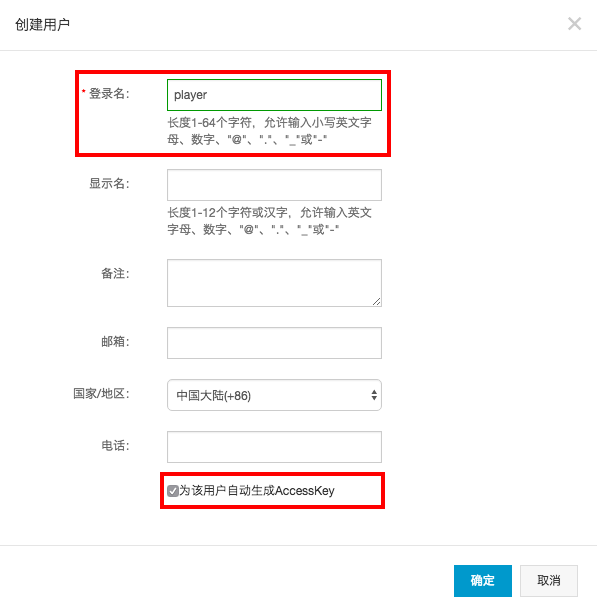
[视频直播服务开通](https://help.aliyun.com/document_detail/29955.html)

3 . 通过访问控制服务创建播放器专用子帐号及其AccessKey：

* a. 登陆[访问控制服务控制台](https://ram.console.aliyun.com/" \l "/overview)
* b. 在用户管理中新建用户：



注意勾选为该用户自动生成AccessKey 选项：



创建子帐号成功，注意保存好该帐号的AccessKey：



* c. 为子帐号分配调用播放器权限：

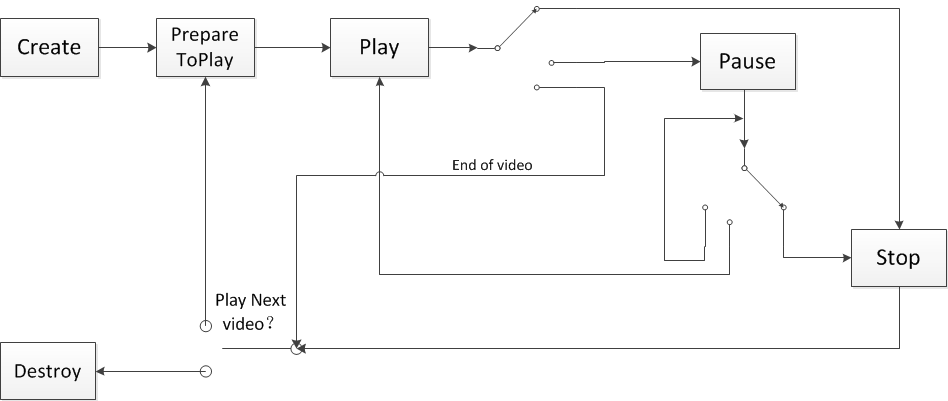
点击**授权**链接：



在**可选授权策略名称**中搜索mts，将**AliyunMTSPlayerAuth**授予此子帐号：



**系统框图**

在开发之前，我们先来了解一下组成播放器的基本模块以及播放器的工作流程，见下图：

**开发步骤**

首先，需要在安卓应用程序中，声明以下权限：

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" /><uses-permission android:name="android.permission.INTERNET" /><uses-permission android:name="android.permission.WAKE\_LOCK"/><uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"/><uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/><uses-permission android:name="android.permission.MODIFY\_AUDIO\_SETTINGS"/>

其次，按照下面的步骤，使用sdk进行播放器的开发：

1. 创建AliVcMediaPlayer播放接口。
2. 注册事件通知函数。
3. 设置缺省解码方式：如果缺省为硬解，会尝试使用硬解，如果失败使用软解；如果缺省为软解，那么会一直使用软解。
4. 如果从历史起点播放，那么调用seek方法。
5. 调用prepareAndPlay准备开始播放。

**demo示例**

在SDK中提供了Demo，此Demo是用播放器SDK开发了一个完整的视频播放器，用户可以参考Demo进行播放器的开发。

下面给出了部分重要的Demo中调用SDK的代码。

一、应用启动的时候，给播放器类执行初始化工作

1. AliVcMediaPlayer.init(getApplicationContext(), businessId, new AccessKeyCallback() {
2. public AccessKey getAccessToken() {
3. return new AccessKey(accessKeyId, accessKeySecret);
4. }
5. });

二、创建播放器，准备视频播放：

1. //1. 创建播放器
2. mPlayer = new AliVcMediaPlayer(context,surfaceView);
3. //2. 注册事件通知
4. mPlayer.setPreparedListener(new VideoPreparedListener());
5. mPlayer.setErrorListener(new VideoErrorListener());
6. mPlayer.setInfoListener(new VideoInfolistener());
7. mPlayer.setSeekCompleteListener(new VideoSeekCompletelistener());
8. mPlayer.setCompletedListener(new VideoCompletelistener());
9. mPlayer.setVideoSizeChangeListener(new VideoSizeChangelistener());
10. mPlayer.setBufferingUpdateListener(new VideoBufferUpdatelistener());
11. //3. 设置缺省编码类型：0表示硬解；1表示软解；
12. mPlayer.setDefaultDecoder(0);
13. //4. 如果从历史点开始播放
14. mPlayer.seekTo(position);
15. //5. 准备开始播放
16. mPlayer.prepareAndPlay(msURI.toString());

三、准备完成事件通知中：

1. private class VideoPrepareListener implements AliVcMediaPlayer.MediaPlayerPreparedListener{
2. @Override
3. public void onPrepared() {
4. //更新视频总进度
5. } }

四、错误事件通知中：

1. private class VideoErrorListener implements AliVcMediaPlayer.MediaPlayerErrorListener {
2. public void onError(int what, int extra) {
3. switch(what)
4. {
5. case MediaPlayer.ALIVC\_ERR\_ILLEGALSTATUS:
6. // 非法状态！
7. break;
8. case MediaPlayer.ALIVC\_ERR\_NO\_NETWORK:
9. //report\_error("视频资源或网络不可用！", true);
10. break;
11. case MediaPlayer.ALIVC\_ERR\_INVALID\_INPUTFILE:
12. //视频资源或网络不可用！
13. break;
14. case MediaPlayer.ALIVC\_ERR\_NO\_SUPPORT\_CODEC:
15. //无支持的解码器!
16. break;
17. case MediaPlayer.ALIVC\_ERR\_FUNCTION\_DENY:
18. //无此操作权限!
19. break;
20. case MediaPlayer.ALIVC\_ERR\_UNKNOWN:
21. //未知错误!
22. break;
23. case MediaPlayer.ALIVC\_ERR\_NOTAUTH:
24. //未鉴权!
25. break;
26. case MediaPlayer.ALIVC\_ERR\_READD:
27. //资源访问失败!
28. break;
29. default:
30. //播放器错误!
31. break;
32. }
33. }
34. }

五、播放信息事件通知中：

1. private class VideoInfolistener implements AliVcMediaPlayer.MediaPlayerInfoListener {
2. public int onInfo(int what, int extra){
3. switch (what)
4. {
5. case MediaPlayer.MEDIA\_INFO\_UNKNOW:
6. // 未知
7. break;
8. case MediaPlayer.MEDIA\_INFO\_BUFFERING\_START:
9. // 开始缓冲
10. break;
11. case MediaPlayer.MEDIA\_INFO\_BUFFERING\_END:
12. // 结束缓冲
13. break;
14. case MediaPlayer.MEDIA\_INFO\_VIDEO\_RENDERING\_START:
15. // 首帧显示时间
16. break;
17. }
18. return 0;
19. }
20. }

**播放器可配置参数与可选功能**

| **配置参数接口** | **用途描述** |
| --- | --- |
| setTimeout | 设置网络超时断开链接的时间 |
| setMaxBufferDuration | 设置直播过程中缓冲区视频丢帧的起始时间，若缓冲区中视频帧的时长超过这个值，则开始丢帧操作。设置这个参数可以控制直播延时的长度，参数值越小则直播的延迟越小。 |

seek功能

| **接口名称** | **用途描述** |
| --- | --- |
| seekTo | seek到指定位置之前的最近的一个关键帧 |
| seekToAccurate | 精准跳转到指定位置 |

除了上述可配置的功能和参数，AlivcMediaPlayer还定义了播放器的事件状态通知和错误代码，以方便开发者掌握播放器的运行状态。

若需要了解上述功能和接口的详细用法，请参照下节的接口说明。

**接口说明**

SDK中提供了两个类AliVcMediaPlayer和AliVcMediaPlayerFactory，其中AliVcMediaPlayer是播放器SDK使用类，AliVcMediaPlayerFactory用来创建播放器AliVcMediaPlayer。同时我们还提供了多个事件通知接口，用来监听播放器的各种状态。

| **名称** | **功能描述** |
| --- | --- |
| AliVcMediaPlayerFactory | 创建媒体播放器接口 |
| MediaPlayer | 媒体播放器功能接口类 |
| AliVcMediaPlayer | 媒体播放器功能实现类 |
| MediaPlayerPrepareListener | 视频播放准备完成监听接口 |
| MediaPlayerCompletedListener | 视频播放完成监听接口 |
| MediaPlayerInfoListener | 视频播放信息监听接口 |
| MediaPlayerSeekCompleteListener | 视频跳转完成监听接口 |
| MediaPlayerBufferingUpdateListener | 视频缓冲监听接口 |
| MediaPlayerVideoSizeChangeListener | 视频大小改变监听接口 |
| MediaPlayerErrorListener | 视频播放错误监听接口 |

**AliVcMediaPlayerFactory**

类名：AliVcMediaPlayerFactory

功能：创建媒体播放器接口类 MediaPlayer

成员：

| **成员** | **功能** |
| --- | --- |
| createPlayer | 创建媒体播放器MediaPlayer |

详细说明：

MediaPlayer createPlayer(Context context,int decoder\_type, String path)

createPlayer用来创建播放器，返回MediaPlayer类。

参数:

• path：播放器文件路径，本地或者网络地址。

返回值：返回空为错误，正确则为有效的MediaPlayer值。

**MediaPlayer**

类名：MediaPlayer

功能：媒体播放器接口类，提供播放控制

成员：

| **成员** | **功能** |
| --- | --- |
| init | 初始化播放器 |
| prepareToPlay | 准备视频播放 |
| play | 开始播放视频 |
| pause | 暂停视频播放 |
| stop | 停止视频播放 |
| reset | 释放播放器 |
| seekTo | 跳转到指定位置 |
| isPlaying | 是否正在播放 |
| setVolumn | 调节音量 |
| getVideoWidth | 获取视频宽度 |
| getVideoHeight | 获取视频高度 |
| getDuration | 获取视频长度 |
| getCurrentPosition | 获取当前视频播放位置 |
| getUserPriority | 获取播放器权限 |
| setPreparedListener | 注册视频准备完成通知 |
| setCompletedListener | 注册播放完成通知 |
| setInfoListener | 注册播放信息通知 |
| setErrorListener | 注册播放错误通知 |
| setSeekCompleteListener | 注册跳转完成通知 |
| setBufferingUpdateListener | 注册缓冲更新通知 |
| setVideoSizeChangeListener | 注册视频大小改变通知 |
| getErrorCode | 获取错误码 |
| setSurfaceChanged | 设置 surface 发生改变 |
| enalbeNativeLog | 打开底层日志，在开发阶段使用 |
| disableNativeLog | 关闭底层日志，在 release 阶段使用 |
| setVideoSurface | 设置视频显示的 surface |
| releaseVideoSurface | 释放视频显示的 surface |
| setTimeout | 设置 IO 超时时间，单位毫秒 |
| setMaxBufferDuration | 设置最大的缓冲时长，直播中有效 |
| setMediaType | 设置视频源类型 |
| setDefaultDecoder | 设置默认的解码器 |
| getPropertyDouble | 获取性能参数 |
| getPropertyLong | 获取长整型性能参数 |
| getCurrNativeLog | 获取 Natvie 的日志 |
| getAllDebugInfo | 获取全部 debug 信息 |
| destroy | 回收播放器 |
| setMuteMode | 设置静音模式 |
| setVideoScalingMode | 设置视频渲染的缩放模式 |

下面详细介绍一下各个成员函数的具体使用：

**init**

1. public static void init(Context context, String
2. businessId, AccessKeyCallback callback);

功能：初始化播放器

参数：

context: Android 上下文；

callback：AccessKey 的回调函数；

businesssId：业务ID，用户自行设置，用于标识使用播放器sdk的APP。如“淘宝直播”就设置“TaobaoLive”。

**prepareToPlay**

public void prepareToPlay(String url);

功能：根据视频文件内容初始化播放器实例，包括读取视频头，解析视频和音频信息，并根据视频和音频信息初始化解码器，创建下载（或读取本地文件）、解码、渲染线程等。

参数：

url：当前播放视频的文件名或网络地址。

**play**

public void play();

功能：播放当前视频。

**pause**

public void pause();

功能：暂停视频播放。

**stop**

public void stop();

功能：停止视频播放。

**destroy**

public void destroy();

功能：回收播放器。

备注：当整个播放器退出时调用，回收播放器。

**reset**

public void reset();

功能：重置播放器。当播放的过程中调用该函数，会先停止当前的播放行为，销毁当前的播放器，然后创建一个新的播放器。

**seekTo**

public void seekTo(int msc);

功能：跳转到指定位置前的第一个关键帧的位置。

参数：msc：跳转的位置，单位为毫秒。

备注：该函数仅允许在点播或播放本地视频过程中调用（直播禁用）。调用后视频会跳转到指定位置前最近的一个关键帧。参数的范围为[0,duration]（duration为视频的时长）。如果传入的参数小于0，则播放器会自动将该参数修正到0；如果传入参数大于duration，则修正到duration。

**isPlaying**

public boolean isPlaying();

功能：视频是否在播放。

返回值：true 代表正在播放，否则没有在播放。

**setVolume**

public void setVolume(int vol);

功能：设置播放器音量

参数：

vol： 音量大小，范围为 0-100，100 为最大，0 为最小。

**getVideoWidth**

public int getVideoWidth();

功能：获取视频宽度

返回值：返回视频宽度。返回值为0表示获取视频宽度失败。

**getVideoHeight**

public int getVideoHeight();

功能：获取视频高度。

返回值：返回视频高度。返回值为0表示获取视频高度失败。

**getDuration**

public int getDuration();

功能：获取视频时长。

返回值：返回视频时长，单位为毫秒。

**getCurrentPosition**

public int getCurrentPosition();

功能：获取视频的当前播放位置。

返回值：视频的当前播放位置，单位为毫秒。

**getErrorCode**

public int getErrorCode();

功能：当播放器出错时，调用该函数获取播放器错误码。

返回值：播放器的错误码。

**setSurfaceChanged**

public void setSurfaceChanged();

功能：通知 surface 改变

备注：在播放暂停或卡顿时，这个时候旋转手机屏幕，会发生渲染错位。为了解决这一问题，请在surfaceChanged发生时，调用此方法。如果播放界面关闭了自动旋转功能，无须调用此方法。

**enableNativeLog**

public void enableNativeLog();

功能：打开底层日志。备注：仅在开发阶段调用此方法。打开底层日志，意味着底层的日志首先会通过adb logcat的形式输出，另外在应用层，还可以通过getCurrNatvieLog方法获取底层日志。

**disableNativeLog**

public void disableNativeLog();

功能：关闭底层日志。

**getCurrNatvieLog**

public List<VideoNativeLog> getCurrNatvieLog();

功能：获取 Native 日志。

备注：仅仅在 enableNativeLog 有效。此方法返回底层日志列表，每条日志(VideoNativeLog) 包含如下几个字段：

Tag：日志的 tag，不唯一。

Content：日志的内容。

Time：日志的时间。

Level：日志的级别（0 表示 ANDROID*LOG* UNKNOWN ； 1 表示 ANDROID*LOG* DEFAULT ； 2 表示 ANDROID*LOG* VERBOSE ；3 表示 ANDROID*LOG* DEBUG；4 表示 ANDROID*LOG* INFO；5 表示ANDROID*LOG* WARN ； 6 表 示 ANDROID*LOG* ERROR ； 7 表 示 ANDROID*LOG* FATAL ； 8 表 示ANDROID*LOG* SILENT）。

**setVideoSurface**

public void setVideoSurface(Surface surface);

功能：设置视频播放 Surface

参数：

surface: 视频播放 View 的 surface。

备注：使用场景是之前的 surface 已经销毁，但是还要继续播放；或者想在一个新的 surface 上显示视频 。特别注意，在初始化播放器的时候，已经传入了 surface，所以在释放以前的 surface 之前，是不允许再次设置新的 surface 的。也就是说请先 releaseVideoSurface 再 setVideoSurface。

**releaseVideoSurface**

public void releaseVideoSurface();

功能：释放视频 Surface。

备注：使用场景是当前的 surface 被销毁；或者想在一个新 surface 上显示视频，需要提前释放当前的 surface。如果使用播放器构造函数或者setVideoSurface 设置了 surface，那么就可以通过releaseVideoSurface 释放当前的 surface，但是一旦释放之后，就不能再次调用，否则就会出现黑屏。

**setTimeout**

public void setTimeout(int timeout);

功能：设置 IO 超时时间。

参数：

timeout：超时时长，单位毫秒。

备注：当播放器超过设定时间没有下载到任何数据，会发送。ALIVC*ERR* LOADING\_ TIMEOUT 错误事件。系统默认 timeout 时间为 15000 毫秒。

**setMaxBufferDuration**

public void setMaxBufferDuration(int duration);

功能：设置直播最大缓冲时长。

参数：

duration: 缓冲时长，单位毫秒。

备注：该函数仅对直播场景有效，主要用于缩短主播与观众之间的时间延迟。当缓冲区中的视频时长超过设置的 duration 时，播放器会自动丢弃部分音视频数据，以减少延迟。系统默认 rtmp、http flv 直播时 duration为8秒，HLS直播时duration为40秒。建议rtmp、http flv直播时，duration 的值超过 GOP 时长（即两个关键帧之间的时间长度）的 2 倍；HLS 直播时，duration 的值超过 m3u8 文件中所有 ts 分片的总时长。这样可以避免出现经常性的视频丢帧。

**setMediaType**

public void setMediaType(MediaType type);

功能：设置播放类型。

参数：

type: 媒体类型。MediaType.Live 表示直播；MediaType.Vod 表示点播

备注：建议在可以清晰分辨视频类型的情况下尽量调用该函数。如果不调用，则播放器会自动根据视频的 duration 来判断媒体类型。Duration 为 0 且格式为 HLS、rtmp、http flv 的为直播类视频，其他为点播类视频。

**setDefaultDecoder**

1. public void setDefaultDecoder(int decoderType);`

功能：设置默认的解码器。

参数：

type: 解码器类型。0代表硬件解码器；1代表软件解码器。

备注：默认为软件解码。由于android手机硬件适配性的问题，很多android手机的硬件解码会有问题，所以，我们建议尽量使用软件解码。

**setMuteMode**

1. public void setMuteMode(boolean on);

功能：设置播放器是否静音。

备注：静音指播放器的静音，并不会影响系统音量。

**setScalingMode**

1. public void setVideoScalingMode(VideoScalingMode scalingMode);
2. enum VideoScalingMode
3. {
4. VIDEO\_SCALING\_MODE\_SCALE\_TO\_FIT(0),
5. VIDEO\_SCALING\_MODE\_SCALE\_TO\_FIT\_WITH\_CROPPING(1)
6. };

功能：设置播放器渲染时的缩放模式，目前有两种模式，VIDEO*SCALING* MODE*SCALE* TO*FIT：等比例缩放显示，如果视频长宽比和屏幕长宽比不一致时，会存在黑边；VIDEO*SCALING*MODE* SCALE*TO* FIT*WITH* CROPPING：带裁边的等比例缩放，如果视频长宽比和屏幕长宽比不一致时，会进行裁边处理以保持全屏显示。

备注：默认为VIDEO*SCALING* MODE*SCALE* TO*FIT* WITH\_ CROPPING模式，可以动态改变。

**getPropertyDouble**

public double getPropertyDouble(int key,double defaultValue);

功能：获取 Double 型性能参数。

参数：

defaultValue：缺省数据。

key：关键字常量。

| **关键字** | **描述** |
| --- | --- |
| MediaPlayer.PROP*DOUBLE* VIDEO*DECODE* FRAMES*PER* SECOND | 视频解码帧率 |
| MediaPlayer.PROP*DOUBLE* VIDEO*OUTPUT* FRAMES*PER* SECOND | 视频渲染帧率 |
| MediaPlayer.FFP*PROP* DOUBLE*OPEN* FORMAT\_ TIME | 调用 avformat*open* input 的时刻，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*FIND* STREAM\_ TIME | 调用 avformat*find* stream\_ info 的时刻，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*OPEN* STREAM\_ TIME | 从开始下载到渲染出第一个视频帧的时间，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*1st* VFRAME*SHOW* TIME | 首个视频帧渲染时刻，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*1st* AFRAME*SHOW* TIME | 首个音频帧渲染时刻，单位毫秒 |
| MediaPlyer.FFP*PROP* DOUBLE*1st* VPKT*GET* TIME | 首个视频帧下载时刻，单位毫秒 |
| MediaPLyer.FFP*PROP* DOUBLE*1st* APKT*GET* TIME | 首个音频帧下载时刻，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*1st* VDECODE\_ TIME | 首个视频帧解码时刻，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*1st* ADECODE\_ TIME | 首个音频帧解码时刻，单位毫秒 |
| MediaPLyaer.FFP*PROP* DOUBLE*DECODE* TYPE | 视频解码器类型 |
| MediaPlayer.FFP*PROP* DOUBLE*LIVE* DISCARD\_ DURATION | 直播视频丢弃音视频帧时长，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*LIVE* DISCARD\_ CNT | 直播视频丢弃音视频帧数量总和 |
| MediaPlayer.FFP*PROP* DOUBLE*DISCARD* VFRAME\_ CNT | 直播视频丢弃视频帧数量总和 |
| MediaPlayer.FFP*PROP* DOUBLE*RTMP* OPEN\_ DURATION | RTMP 流打开时长，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*RTMP* OPEN\_ RTYCNT | RTMP 重连次数 |
| MediaPlayer.FFP*PROP* DOUBLE*RTMP* NEGOTIATION\_ DURATION | RTMP 连接握手时长，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*HTTP* OPEN\_ DURATION | HTTP 流打开时长，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*HTTP* OPEN\_ RTYCNT | HTTP 重连次数 |
| MediaPlayer.FFP*PROP* DOUBLE*HTTP* REDIRECT\_ CNT | HTTP 重定向次数 |
| MediaPlayer.FFP*PROP* DOUBLE*TCP* CONNECT\_ TIME | TCP 连接时长，单位毫秒 |
| MediaPlayer.FFP*PROP* DOUBLE*TCP* DNS\_ TIME | TCP 连接 DNS 时长，单位毫秒 |

**getPropertyLong**

public long getPropertyLong(int key,long defaultValue);

功能：获取 Long 型性能参数。

参数：

defaultValut：缺省数据。

key：关键字常量。

| **关键字** | **描述** |
| --- | --- |
| MediaPlayer.FFP*PROP* INT64*VIDEO* CACHED\_ DURATION | 视频缓冲时长，单位毫秒 |
| MediaPlayer.FFP*PROP* INT64*AUDIO* CACHED\_ DURATION | 音频缓冲时长，单位毫秒 |
| MediaPlayer.FFP*PROP* INT64*VIDEO* CACHED\_ BYTES | 视频缓冲大小，单位 byte |
| MediaPlayer.FFP*PROP* INT64*AUDIO* CACHED\_ BYTES | 音频缓冲大小，单位毫秒 |
| MediaPlayer.FFP*PROP* INT64*VIDEO* CACHED\_ PACKETS | 视频缓冲帧数 |
| MediaPlayer.FFP*PROP* INT64*AUDIO* CACHED\_ PACKETS | 音频缓冲帧数 |
| MediaPlayer.FFP*PROP* INT64*SELECTED* VIDEO\_ STREAM | 视频流 index |
| MediaPlayer.FFP*PROP* INT64*SELECTED* AUDIO\_ STREAM | 音频流 index |

**getAllDebugInfo**

public Map<String, String> getAllDebugInfo();

功能：获取实时性能数据。

备注：返回的性能参数包含：

1. "dec-fps"：视频解码 fps
2. "out-fps"：视频渲染 fps
3. "select-v"：视频流 index
4. "select\_a"：音频流 index
5. "v-dec"：视频解码器名称
6. "a-dec"：音频解码器名称
7. "vcache-dur"：视频缓冲时长，单位秒
8. "acache-dur"：音频缓冲时长，单位秒
9. "vcache-bytes"：视频缓冲大小，单位 byte
10. "acache-bytes"：音频缓冲大小，单位 byte
11. "vcache-pkts"：视频缓冲帧数
12. "acache-pkts"：音频缓冲帧数

**AliVcMediaPlayer**

MediaPlayer的实现类。

**MediaPlayerPrepareListener**

当调用prepareAsync后,视频准备完成后会发送准备完成事件，用户需要注册该事件，以便获取到该事件通知，在准备完成后调用start接口进行视频播放。

1. public interface MediaPlayerPrepareListener {
2. void onPrepared();
3. }

**MediaPlayerCompletedListener**

当视频播放完成后，会发出该事件通知消息，用户需要注册该事件，在播放完成后完成相关清理工作。

1. public interface MediaPlayerCompletedListener {
2. void onCompleted();
3. }

**MediaPlayerInfoListener**

当视频开始播放，用户需要知道视频的相关信息，可以注册该事件。

1. public interface MediaPlayerInfoListener{
2. void onInfo(int what, int extra);
3. }

参数：

what：获取到的播放信息或警告的类型.播放的相关信息有：

• MEDIA*INFO* UNKNOW：未知信息

• MEDIA*INFO* BUFFERING\_ START：当开始缓冲时，收到该信息

• MEDIA*INFO* BUFFERING\_ END：缓冲结束时收到该信息

extra：对播放信息的额外表述。

**MediaPlayerErrorListener**

当视频播放出现错误后,会发出该事件通知消息，用户需要注册该事件通知，以便在出现错误后给出相关错误提示。

1. public interface MediaPlayerErrorListener {
2. void onError(int what, int extra);
3. }

参数：

what：错误信息的类型.错误信息有：

• ALIVC*ERR* UNKNOW：未知错误

• ALIVC*ERR* LOADING\_ TIMEOUT：缓冲超时

• ALIVC*ERR* NO\_ INPUTFILE：未设置视频源

• ALIVC*ERR* NO\_ VIEW：无效的surface

• ALIVC*ERR* INVALID*INVALID* INPUTFILE: 无效的视频源

• ALIVC*ERR* NO*SUPPORT* CODEC：无支持的解码器

• ALIVC*ERR* FUNCTION\_ DENIED：操作无权限

• ALIVC*ERR* NO\_ NETWORK: 网络不可用

• ALIVC*ERR* ILLEGALSTATUS：非法状态

• ALIVC*ERR* NOTAUTH：未鉴权

• ALIVC*ERR* READD：视频源访问失败

extra：错误信息的额外描述

• ALIVC*ERR* EXRA\_ DEFAULT：缺省值

• ALIVC*ERR* EXTRA*PREPARE* FAILED：prepare失败

• ALIVC*ERR* EXTRA*OPEN* FAILED: open stream 失败

**MediaPlayerSeekCompleteListener**

当视频进行seek跳转后，会发出该事件通知消息，用户注册该事件通知后，能收到跳转完成通知。

1. public interface MediaPlayerSeekCompleteListener {
2. void onSeekCompleted();
3. }

**MediaPlayerBufferingUpdateListener**

当网络下载速度较慢来不及播放时，会发送下载缓冲进度通知。

1. public interface MediaPlayerBufferingUpdateListener {
2. void onBufferingUpdateListener(int percent);
3. }

参数:

percent: 目前视频缓冲的进度，范围为0-100，100代表缓冲完成，0代表缓冲开始。

**MediaPlayerVideoSizeChangeListener**

当视频播放时视频大小改变后，会发出该事件通知。

1. public interface MediaPlayerVideoSizeChangeListener {
2. void onVideoSizeChange(int width, int height);
3. }

参数：

width：视频改变之后的宽度

height：视频改变之后的高度

**版本更新说明**

1.0 原始版本，调用硬件的mediaPlayer接口实现

2.0 改用硬件的mediaCodec接口实现

2.1 增加直播秒开功能、缓冲区丢帧策略等。

2.2 支持多实例、支持https、增加静音功能、添加视频渲染时的缩放模式